Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

До захисту допущено

Завідувач кафедри

_____ _____
(підпис) (ініціали, прізвище)
«___» _____ 2019 р.

Дипломна робота

Освітньо-кваліфікаційного рівня «бакалавр»
з напряму підготовки (спеціальності) **6.050102 «Комп'ютерна інженерія»**
на тему
«Програмна реалізація настільної гри»

Виконав: студент 4 курсу, групи ІО-53 Дівак Олексій Павлович _____
(підпис)
Керівник: ст. викл. Алещенко Олексій Вадимович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)
Консультант: нормконтроль проф. Сімоненко Валерій Павлович _____
(назва розділу) (вчене звання, науковий ступінь, прізвище та ініціали) (підпис)
Рецензент: _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що в цьому дипломному
проекті немає запозичень з праць
інших авторів без відповідних посилань
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність Комп'ютерні системи та мережі

ЗАТВЕРЖДУЮ

Завідувач кафедри

_____ _____

(підпис) (ініціали, прізвище)

«___» _____ 2019 р.

ЗАВДАННЯ

на бакалаврську дипломну роботу студента

Дівака Олексія Павловича

(прізвище, ім'я, по-батькові)

1. Тема проекту (роботи) *Програмна реалізація настільної гри*

Керівник проекту (роботи) *ст. викл. Алещенко Олексій Вадимович*

(прізвище, ім'я, по-батькові)

затверджені наказом по університету від «23» квітня 2019 року №1180-С

2. Термін здачі студентов закінченого проекту (роботи) *20 травня 2019р.*

3. Вихідні дані до проекту (роботи) *технічна та документація, теоретичні та статистичні дані*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються) *Опис предметної області, аналіз проблем під час реалізації, розробка архітектури, розробка гри*

5. Перелік графічного матеріалу (з точним позначення обов'язкових креслень) структурна схема системи, узагальнена схема роботи системи, блок-схема алгоритму модуля редактора, схема ____ інформаційних вікон редактора

6. Консультанта проекту (робота), з вказівкою розділів роботи, які до них відносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проектування (роботи)	Строк виконання етапів проекту (роботи)	Примітки
1.	Вивчення літератури за тематикою роботи	17.11.2019	
2.	Розроблення та узгодження технічного завдання	30.11.2019	
3.	Аналіз існуючих імплементацій	14.02.2019	
4.	Вибір мови програмування	23.02.2019	
5.	Розгляд типів вибраної мови програмування	01.03.2019	
6.	Програмна реалізація типів	05.04.2019	
7.	Програмна реалізація ефектів	20.04.2019	
8.	Підготовка матеріалів текстової частини проекту	20.05.2019	

Студент-дипломник _____
(підпис)

Керівник роботи _____
(підпис)

Анотація

Дана бакалаврська дипломна робота розглядає існуючі на даний час стратегії програмної побудови основи настільної гри, зокрема карткової настільної гри, і, використовуючи поняття чистого функціонального програмування, формулює інший підхід до предмету в формі проміжного інтерфейсу. Сам рівень — за умови наявності відповідних імплементацій як для сервера, так і клієнта — описує більш безпечний підхід, заснований на розрахунку стану з обох сторін для забезпечення достовірності, з побічним ефектом скорочення обсягу даних, що передаються. Програма написана на мові Haskell, що дозволяє знизити складність обчислень за допомогою вбудованої мемоізації і зробити програму більш безпечної на рівні типів, забороняючи певні типи взаємодій за визначенням.

Аннотация

Данная бакалаврская дипломная работа рассматривает существующие в настоящее время стратегии програмного построения основы настольной игры, в частности карточной настольной игры, и, используя понятия чистого функционального программирования, формулирует другой подход в форме промежуточного интерфейса. Этот интерфейс — при условии наличия соответствующих имплементаций со стороны как сервера, так и клиента — описывает более безопасный подход, основанный на расчете состояния с обеих сторон, для обеспечения достоверности, с побочным эффектом сокращения объема передаваемых данных. Программа написана на языке Haskell, что позволяет снизить сложность вычислений с помощью встроенной мемоизации и сделать программу более безопасной на уровне типов, запрещая определённые типы взаимодействий по определению.

Abstract

This Bachelor's thesis takes a glance at the currently existing strategies of programmatically building a foundation for a tabletop game, particularly a board game, and, by using the notions of pure functional programming, formulates a different approach to the subject in the form of an intermediate stateful interface layer. The layer itself — provided the matching instances for the server and the client exist — emphasizes a safer approach, based on calculating state on both sides for ensuring validity, with a side effect of reducing the amounts of data that is needed to be transferred. The program is written in Haskell, allowing for the higher computational complexity to be mitigated with the use of in-built memoization, as well as adding a level of type safety, disallowing certain levels of interactions by definition.

Зміст

Розділ 1.
Опис предметної області

1.1 Загальний огляд настільних ігор

Відповідно до визначення настільні ігри (англ. tabletop games) – ігри, у які грають на столі або на будь-якій іншій пласкій поверхні. Це дещо абстрактне визначення об’єднує широкий пласт ігор, і поділяється на наступні категорії:

- Пригодницькі ігри (англ. adventure games) – ігри, в якій гравець бере на себе роль протагоніста у деякому сюжеті, що базується на розв’язуванні загадок та відкритті нових знань.
- Ігри на спеціальній дошці (англ. board games) – ігри, що базуються на переміщенні фігур на спеціальній дошці, при цьому наявний деяких набір правил. Незважаючи на це, у деяких іграх (наприклад Dungeons & Dragons) дошка служить тільки для візуалізації сценарія гри, а не як рушійна сила самої гри. Варто зауважити, що і “tabletop games”, і “board games” можно перевести на українську мову як настільні ігри, але в контексті цієї роботи це неважливо.
- Карткові ігри (англ. card games) – ігри, у якій гральні карти є головним єлементом гри. Карти у цьому розумінні можуть бути традиційні (36/52-картові колоди), або специфічні до самої гри. Всі ігри, що використовують традиційні набори карт (покер, преферанс) належать до цієї категорії.
- Ігри, що використовують гральні кісточки (англ. dice games) – ігри, що використовують гральні кісточки як головний або один з методів отримання випадковості.

- Ігри на папері (англ. pen and pencil games) – ігри, що використовують лист бумаги та олівець. Їх простота забезпечила їх широку розповсюдженість, за рахунок того, що вони не потребують ластику.
- Рольові ігри (англ. role-playing games) – ігри, у яких гравці беруть на себе роль персонажів у видуманому світі і впродовж сеансу грають роль цього самого персонажу.
- Стратегії (англ. strategy games) – ігри, що передбачають високий рівень взаємодії між гравцями, від чого напряму залежить хід гри. Найрозповсюдженіший приклад стратегічної гри: шахи, де обидва гравці мають продумувати ходи одне одного і виграє той, хто продумує більше ходів, ніж інша сторона.
- "Плиткові" ігри (англ. tile-based games) – ігри, що використовують плитки, як базовий єлемент гри.

Оскільки усі з цих підкатерогій теж достатньо абстракні, більшість складних ігор входять до декількой категорій у той самий час, що робить їх класифікацію дещо складною, а більшість ігор, що підходять під неї приналежними до декількох підкатегорій одночасно.

У рамках цієї дипломної роботи мною була вибрана настільна гра Arkham Horror [1] (2nd Edition, 2005) компанії Fantasy Flight Games [2], що базується на всесвіті, створеному Говардом Лавкрафтом, і розповідає історію декількох дослідників (англ. investigator) в 1926му році, що в один невдалий день намагаються зупинити вторгнення могутної інопланетної істоти у їх місто. Гра використовує спеціальну дошку, кожен з гравців грає за одного з дослідниць і базується на великому наборі спеціальних карт, що відповідно до вищезгаданої класифікації робить її і пригодницькою, і грою зі спеціальною дошкою, і картковою.

Оскільки як такі настільні ігри програмно на даний час не виготовляються (і більшість з них просто-напросто портовані до Tapletop Simulator, що є фактично симулятором кімнати, де кожна карта – об'єкт, що можна переміщувати і не має як такового набору правил), у контексті цієї

дипломної роботи будемо розглядати існуючі карткові ігри, адже вони, як і вибрана мною гра, базуються на картках, хоча і в дещо більшій мірі, аніж у моєму випадку.

1.2 Аналіз існуючих рішень

1.2.1 Hearthstone

Код гри Hearthstone [3] є закритим, відповідно, неможливо провести детальний аналіз технік, що використовуються у даній грі. Однак, як найпопулярніша карткова гра на даний момент, багато людей грає у цю гру, що дає можливість оцінити техніку, за якою вона створена, на прикладі різноманітних помилок реалізації цієї гри.

Наприклад, на Рис. 2.1 зображено одну з таких ситуацій: на даному етапі гри було розіграно дві карти та гравець узяв чотири.

Рис 1.1. Ціни карт, що не співпадають, виділені білим

З розіграних карт:

- Scepter of Summoning застосовує до ціни карти наступний ефект:

$$F(Price)=\begin{cases}5, Price\geq 5\\ Price, Price<5\end{cases}$$

- Robes of Gaudiness застосовує до ціни карти наступний ефект:

$$F(Price)=\lfloor Price/2\rfloor$$

Неважко оцінити, що фінальна формула після розігрування Scepter of Summoning, а потім Robes of Gaudiness має мати наступний вигляд:

$$F(Price)=\begin{cases}\lfloor 5/2\rfloor, Price\geq 5\\ \lfloor Price/2\rfloor, Price<5\end{cases}=\begin{cases}2, Price\geq 5\\ \lfloor Price/2\rfloor, Price<5\end{cases}$$

Відповідно Baron Geddon, як карта ціною у сім, має коштувати за цією формулою два, адже сім більше двох.

Незважаючи на це, карта у руці даного гравця коштує три, ймовірно тому, що Scepter of Summoning розігрується на усі карти колоди після вибору перших трьох карт, що, саме собою, виходячи з опису карт, є помилкою розробки даної гри.

Наступний приклад зображений на Рис. 2.2. У ньому:

- гравець розіграв усі карти зі своєї колоди окрім трьох:
  - Reckless Experimenter має ефект, що говорить: усі карти з передсмертним хрипом (ефект, що виконується, коли кількість здоров'я карти падає нижче одиниці) коштують на три менше, але вмирають у кінці ходу;
  - Voodoo Doll, з боєвим кличем (ефект, що виконується, коли карта розігрується), що говорить: вибрати карту-міньона; та передсмертним хрипом, що вбиває вибрану під час розігрування цієї карти карту;
  - Mecha'Thun, з передсмертним хрипом, що знищує опонента, якщо у гравця немає карт ні в колоді, ні у руці, на на столі.
- гравець розігрує Rechkless Experimenter, Voodoo Doll,, обираючи ціллю Reckless Experimenter та Mecha'Thun (саме у такій послідовності).

- гравець завершує свій хід.

Система гри Hearthstone описує, що, якщо одночасно вмирає декілька карт з передсмертними хрипами, то їх ефекти вступають у гру у тій послідовності, у якій були розіграні самі карти, що означає, що мали трапитися:

- Смерть Voodoo Doll, як результат ефекту Reckless Experimenter;
- Смерть Reckless Experimenter, як результат ефекту Voodoo Doll;
- Смерть Mecha'Thun, як результат ефекту Reckless Experimenter, незважаючи на те, що ця карта вже вмерла, адже подібні постійні ефекти діють на усі інші карти, що вмирають одночасно.

Відповідно, оскільки Mecha'Thun є останньою картою, що вмирає у даній послідовності, то гра має завершитися.

Рис. 1.2

Незважаючи на це, на Рис. 2.3 бачимо, що гра продовжується і у історії гри записано, що ефект карти Mecha'Thun дійсно відбувся останнім, що відповідно до логіки гри не має жодного сенсу.

Рис. 1.3 Стан гри після розігрування трьох карт

Таким чином можна зробити висновок, що незважаючи на добру базову імплементацію цієї карткової гри, с плином часу і добавлення більше аніж десяти доповнень до неї, код цієї гри не є оптимальним і фактично не має чітко описаної структури виконання, адже далеко не усі ефекти виконуються так, як мають.

Зауважу також, що ця гра розробляється компанією Blizzard Entertainment, що станом на 2012й рік мала понад 4700 робітників та станом на 2015й рік її чистий прибуток за рік становив більше, аніж 223 мільйона долларів. Незважаючи на величезні прибутки, ця компанія все одно не змогла забезпечити хорошу структуру цієї карткової гри, що в ретроспективі коштує їм набагато більше робочих часів, аніж могло би.

1.2.2 Інші імплементації

На жаль усі інші ігри або мають закритий код і не мають достатньої кількості гравців, або мають відкритий код і фактично нуль гравців, відповідно відслідити способи імплементації відповідних ігор або майже неможливо, або потребує дуже ретельного ознайомлення зі структурою вже існуючих систем, що, зазвичай, погано задокументовані.

Існує також блог [4] (також на рис. 1.1) двох друзів, що вісім років тому назад забажали створити програмну реалізацію цієї самої гри використовуючи C# та XNA Framework[5].

Рис 1.3

Блог сам собою дуже розпливчасто описує процес створення гри, не вникаючи у технічні особливості, але з нього зрозуміло, що:

- автор був програмістом-прочатковцем і почав роботу над проектом для покращення своїх навичок програмування
- автор підійшов до проблеми стандартним ООП-способом (наприклад переміщення слайдерів вмінь реалізоване дуже нелогічно описом нових значень вмінь, як деякого стану, що зберігається між ходами, при цьому гравець може робити до N змін у рамках належного кожен

хід – дуже базова імплементація, що гарантовано мала би бути переписана у майбутньому.

Впродовж половини року і десятка постів у блозі описуються проблеми, з якими стикався автор, і, нарешті, посеред літа 2011-го року блог переривається. У коментарях до останнього поста автор через рік відповів, що він не збирається дообляти гру, адже він продвинувся вже достатньо далеко у своєму вивченні програмування, щоб забути структуру гри і сама гра є дуже великою, твердження, сказане ним декілька разів впродовж усього блогу.

Висновки до розділу 1

У даному розділі мною були розглянуті історія настільних ігор, їх типи та декілька їх комп'ютерних імплементацій, і відповідно було досягнуто висновку, що на даний момент настільні ігри не мають широкого розповсюдження у світі комп'ютерних реалізацій, оскільки складаються з великої кількості елементів та мають достатньо складні набори правил, що не дає можливості розробникам швидко їх створювати.

У той же час дивлячись на вже існуючі реалізації зрозуміло, що методи створення таких програмних продуктів не є ідеальними, адже вони мають багато помилок і погано масштабуються.

Також була осмотрена попередня спроба реалізації цієї самої гри вісім років назад, що в кінці-кінців не мала результату через хаотичність процесу розробки, відсутність досвіду створення автором великих програм та неправильність підходів.

Розділ 2.

Аналіз проблем під час реалізації

2.2 Вибір стилю написання програми

Виходячи із логіки роботи більшості наявних на даних момент карткових ігр і мов, на яких вони написані, стає очевидним, що фактично усі ігри використовують подібну систему побудови гри, що полягає у описі гри відповідно до правил ООП. Таким чином:

• Кожен елемент світу є об’єктом, що має деякий стан і може виконувати деякі дії над іншими об’єктами;

• Коли гравець розігрує карту, виконання дій цілком залежить від самої карти, що означає, що, у цей момент часу, карта має повний контроль над станом гри, обмежений, хіба що, самим програмістом;

• Запис історії у подібній системі спрощується до простого логування подій, що вже відбулися і може не відповідати дійсності;

• Хоча спочатку описання взаємодій дуже просте, що фактично і є пунктом продажу ООП у даному контексті, зі збільшенням кількості ефектів до сотень або тисяч, їх групування та опис взаємодій потребує все більшої ретельності від програміста, що гарантовано приведе до помилок надалі;

• Відсутність дуже чіткої структури програми перетікає у складності реалізації передачі даних між клієнтами, адже різні частини програми можуть виконуватися по-різному

Відповідно можна заключити, що ООП хоча і є достатно привабливою ідеєю для малих карткових ігр і, за умов абсолютно вірного використання і надретельних програмістів, несумнівно дає швидкий, економний відносно використання пам’яті і зручний код, зі збільшенням розміру бази коду підтримка стає експоненціально складнішою.

У той же час існує інший спосіб написання програм, що довгий час залишався у тіні ООП: чисто функціональне програмування. Відповідні ідеї цієї парадигми:

• чітке розділення програмних ефектів на чисті (ті, що не залежать від зовнішнього світу і результат яких завжди однаковий) та нечисті дозволяє кращу компартменталізацію програми та набагато чіткіше розуміння взаємодій на теренах програми;

• ліниві обчислення, як ідея відкладення обчислень до моменту необхідності, що дозволяє економити обчислювальні ресурси. У поганих випадках це призводить до використання жахливо великих розмірів пам’яті, але у той же час ліниві обчислення завжди завершуються у кількість часу рівну, або меншу за такі самі обчислення строго;

• інші структури даних, наприклад зазвичай базовими типами зберігання даних у чисто функціональних мовах програмування є однозв’язні списки та хеш-таблиці, що мають інші параметри відносно звичайних масивів і зазвичай підходять гірше під ідеї імперативних мов програмування.

Відповідно, одна з можливих концепцій написання карткової гри включає в себе зберігання початкового стану усього світу в самому початку гри з подальшим накоплюванням історії світу і прийнятті рішень відповідно до вже існуючої історії. Це дає такі можливості:

• історія усього світу вже наявна і її не треба логувати;

• можливо перевірити правильність вже збереженого світу;

• зберігання і відтворення світів описується усього двома структурами даних, без жодних проміжних станів;

• мови програмування у яких наявна мемоізація (збереження значень проміжних даних) за правильного використання дозволять різко зменшити кількість обчислень;

• чітке описання взаємодій дозволить передавати лише необхідні дані між учасниками гри, економлячи мережевий трафік.

2.2 Аналіз можливостей системи типів мови Haskell

Мова Haskell[6] добре підходить під поставлені завдання, оскільки вона є чисто-функціональною, має вбудовану мемоізацію та потужну систему типів, що дозволить створити зрозумілу, просту та відносно коротку програму.

2.2.1 Базові типи

Типи у мові Haskell описуються наступним чином:

```
data [context =>] type tv1 ... tvi
  = con1 c1t1 c1t2 ... c1tn
  | ...
  | conm cmt1 ... cmtq
    [deriving]
```

, де:

- `[context =>]` описує список класів типів, що зв'язують цей тип і відповідно, не є обов'язковими. Це не вважається важливою частиною мови та у майбутньому буде видалено з мови. Використання цієї частини опису типів на даний момент можливе тільки за умови використання прагми `{-# LANGUAGE DatatypeContexts #-}`;
- `type` є саме назвою конструктора типу даних (англ. data type constructor);
- `tv1 ... tvi` описують будь-які інші типи, що будуть використовуватися у конструкторах типів надалі;
- `con1 | ... | conm` описують типи змінних;
- `c1t1 ... c1tn та cmt1 ... cmtq` описують саме дані, які зберігає відповідний тип змінних;
- `[deriving]` існує для створення класів типів нашвидкоруч. Один з таких класів, Show, дозволяє отримувати текстову репрезентацію даних за умови, що всі дані у типі теж є членами цього класу.

Можливо описати синонім вже існуючого типу фразою

```
type A a .. b = B a .. b
```

при чому будь-який з параметрів `a .. b` справа може бути замінений на деякий тип з винесенням нових параметрів замість створеного вліво.

Також існує альтернативний спосіб опису типів з тільки одним типом даних та одним полем

```
newtype type tv1 .. tvn i
 = constructor value
   [deriving]
```
, де

- `type` збігається з визначенням для типів, створених словом `data`;
- `tv1 ... tvn` збігається з визначенням для типів, створених словом `data`;
- `constructor` є єдиним неопціональним конструктором типів змінних;
- `value` є єдиним неопціональним типом змінних;
- `[deriving]` збігається з визначенням для типів, створених словом `data`;

Істотна відмінність `newtype` від `data` у тому, що `newtype` фактично є більш низькорівневим типом, адже компілятору не треба розрізняти конструктори та типи даних – для даного типу завжди існує лише один.
Фактично усі дані

Ця система типів не була б настільки потужною за відсутності поліморфізму. Наприклад, один з базових типів:

```
data Maybe a = Just a | Nothing
```

може описувати будь-який з:

- `Just 3 :: Maybe Int`
- `Just Nothing :: Maybe (Maybe ())`
- `Nothing :: Maybe Void`

абсолютно нативно і без будь-яких перешкод.

У контексті цієї дипломної роботи поряд з типами сімейств важливо відмітити ще існування фантомних типів. Фантомний тип – тип, що використовується даним типом даних, але жоден з типів змінних його не використовує. Наприклад, у типі

```
data B a = B
```

значення `a` може бути будь-яким, оскільки воно не впливає на значення самого типу змінних. Цей, на перший погляд безглуздий спосіб використання системи типів дає нам додаткові можливості програмування на рівні типів, оскільки `B :: B Bool` та `B :: B ()` не збігаються на рівні типів, незважаючи на те, що самі дані однакові.

2.2.2 Сімейства типів

Сімейства типів у мові програмування Haskell можливо використовувати лише з прагмою `{-# LANGUAGE TypeFamilies #-}`.

Концепція сімейств типів, як і усі інші частини мови Haskell, є частиною теорії типів. Сімейство типів у теорії типів – часткова функція на рівні типів, відповідно, застовування цієї функції стосовно деяких типових параметрів дає тип. Сімейства типів дозволяють программам вираховувать типи даних, на яких програма буде оперувати, а не мати їх статично описаними у контексті програми. Сімейства типів таким чином є для звичайних типів аналогом методів классів для стандартних функцій мови, описуючи не одне визначення, а деяку множину визначень, з яких буде вибраний вірний член, якщо такий існує.

Нехай маємо тип

```
data Scope = Valid | Invalid
```

який описує бінарний вибір (і є фактично ідентичним до опису булевої функції мови Haskell), сімейство типів

```
type family IsValid (a :: Scope) where
    IsValid Valid = 'True
    IsValid _     = 'False
```

та тип

```
data Eff a where
    Bottom :: Eff
    Right :: Eff a ->Eff a
    Wrong :: IsValid a ~ 'False => Eff a -> Eff b
```

Відповідно, вірними типами є

```
Bottom :: Eff a
Right Bottom :: Eff a
Wrong (Bottom :: Eff ‘Invalid)
 :: (IsValid b ~ ‘True)
 => Eff b
```

і у той же час

```
Wrong (Bottom :: Eff ‘Valid)
```

не є вірним типом.

Важливо зазначити, що у контексті мови Haskell твердження

```
Wrong Bottom
```

не має сенсу, оскільки компілятор не підбирає типи навмання, а лише робить підстановку на кожному рівні.

Ця дуже проста система дає деяку можливість задання вірних і невірних типів даних у подібних системах, що описують однонаправлені графи, що фактично робить можливим вилучення частини типів з множини вірних для гарантії вірності коду програми.

2.2.3 Квантор існування

Під час опису типів за допомогою слова `data` можливо оминути згадування використовуваного справа від знаку дорівнює типу зліва від нього. Наприклад, у типі

```
data A = forall a. A a
```

`a` не існує зліва, відповідно компілятор не має змоги відслідити тип цих даних при наданні йому деякого `A`. Ця техніка називається квантор існування (англ. existential quantification) і загалом у таких випадках створюється деяких клас

```
class isA where
```

що дозволяє описати

```
data A = forall a. IsA a => A a,
```

що у свою чергу зменшує список усіх можливих типів схованого типу змінних тільки до примірників відповідного класу типів.

2.2.4 Залежні типи

Залежні типи у майбутньому можливо буде використовувати за допомогою прагми `{-# LANGUAGE DependentHaskell #-}`.

Залежні типи (англ. dependent types) об’єднують типи даних і змінних у одне монолітне поняття типу, що саме собою дає можливість лаконічно описувати структури даних, частини котрих залежать від інших частин цієї самої структури даних. Наприклад, опис кортежу

```
(a :: A k, b :: k)
```

логічно дуже простий: обидві частини кортежу мають мати зв’язані типи, при чому кожна з частин потребує вірності іншої, але на практиці це також передбачає наявність деякої інформації також на стороні самих даних для того, щоб взнати конкретний тип даних (наприклад у прикладі з сімействами типів). Як інструмент, залежні типи будуть імплементовані у мові Haskell не раніше, аніж у 2020-му році, відповідно його використання назараз неможливе.

Варто зауважити, що також існують мови програмування (наприклад Adga), що повноцінно підтримують залежні типи, але вони дуже повільні, оскільки імплементуються поверх інших мов програмування і відповідно достатньо поганий вибір для комп’ютерної гри у контексті швидкості роботи.

Висновок до розділу 2

У даному розділі мною були проаналізовані ідеї, що використовуються під час проектування систем, на яких будуються карткові ігри, та був запропонований інший спосіб реалізації, що грунтується на використанні концепцій чисто функціонального програмування. В ідеалі цей підхід має надати велику кількість переваг, серед яких:

- чітка структура взаємодій в межах програми;
- проста і потужна система опису стану;
- можливе зменшення кількості обчислень за умови правильної імплементації програми;

У той же час вірогідне набагато більш агресивне використання наявної пам’яті комп’ютера, що не має бути проблемою надалі, адже ОЗП стає все більш дешевою і компактною.

Також був проведений екскурс у систему типів мови Haskell з перехідом до поглиблених рівнів, що надалі допоможе у створенні програми, що є більш безпечною на рівні типів.

Розділ 3.
Розробка архітектури

3.1 Базова механіка

Одна з найважливіших базових механік гри Arkham Horror: перевірка вмінь (англ. check**)** – фактично описує підкидання деякої кількості кісток і підрахунок кількості “вдалих” результатів.

У загальному випадку це має вигляд **[skill] check [modifier] [int]**, де:

- **[skill] –** один з десяти типів вмінь, наявних у грі, що вираховуються зі стану дослідника гравця, що виконує підбрасування костей. З цих вмінь шість базових:
  - Speed – уособлює швидкість реакції дослідника;
  - Sneak – уособлює вміння дослідника пересуватися непомітно;
  - Fight – уособлює вміння дослідника битися;
  - Will – уособлює духовну силу;
  - Lore – уособлює знання, що має цей дослідник;
  - Luck – уособлює удачу дослідника;

  і чотири розвинених:
  - Combat – базується на Fight, плюс наявна зброя, магічна або стандартна;
  - Evade – базується на Sneak, описує вмісння дослідника уникати бій;
  - Horror – базується на Will, описує можливість дослідника морально протистояти мерзотності ворогів;
  - Spell – базується на Lore, описує вміння працювати з магією;
- **[modifier] –** функція зміни кількості кісточок, завжди у формі (+n) або (-n)
- **[int]** – важкість перевірки, якщо фінальна кількість “вдалих” кісточок менша за це значення, то перевірка вважається проваленою.

При цьому “вдалою” кісткою вважаються:

- у загальному випадку кістка зі значенням 5 або 6;
- якщо дослідник проклятий, то тільки кістки зі значенням 6;
- якщо дослідник благословенний, то кістки зі значеннями 4, 5 або 6.

Кожен дослідник має деяку кількість розсудливості та деяку кількість витривалості. Якщо дослідник витрачає усю:

- витривалість, то він/вона переміщується до локації St. Mary’s Hospital;
- розсудливість, то він/вона переміщується до локації South Church;
- і витривалість, і розсудливість, то дослідник вважается таким, що був зжертим, і гравець має вибрати нового дослідника з колоди дослідників.

Гра поділяється на ходи, кожен з яких ділиться на фази. Кожну фазу перший гравець виконує свою частину фази, а потім по колу усі інші. В кінці ходу перший гравець передає свій маркер наступному гравцю у колі. Відповідно, фази;

- Upkeep – усі дослідники можуть передвинути свої повзунки відповідно до своєї сфокусованості та всі карти, які граються у цю фазу мають бути зіграні.
- Movement – кожен дослідник переміщається по місту на кількість переходів, що відповідає його/її кількості очків переміщення. При цьому дослідник має або ухилятися від атак чудовиськ, якщо вони наявні на місці, у яке бажає передвинутися дослідник, або битися з цими самими чудовиськами.
- Arkham Encounters – кожен дослідник у місті або обирає спеціальну пригоду на своєму місцезнаходженні або бере одну з колоди пригод для цієї локації, якщо він/вона знаходиться у локації.
- Other World Encounters – кожен дослідник у іншому світі бере карти з колоди пригод у іншому світі до тих пір, поки він/вона не знайде карту

того кольору, що відповідає кольору того світу, у якому вони знаходяться.

- Mythos Phase – перший гравець бере карту міфу, переміщує усіх монстрів у місті підповідно то типу відповідного монстра і карти міфу, відкриває новий портал та виконує дії, вказані на карті.

3.2 Типи елементів

Arkham Horror, у своїй базовій версії складається з таких елементів:

- Дошка (зображена на рис 3.1) , що в свою чергу складається з:
  - дев’яти районів: Downtown, Easttown, French Hill, Merchant District, Miskatonic University, Northside, Rivertown, Southside, Uptown;
  - дев’яти вулиць, кожна з яких належить одному з районів. Вулиці з’єднані одна з одною у порядку близості по два-три;
  - 26 відкритих локацій, кожна з яких належить одному з районів і має вихід до вулиці того ж району. Кожна з цих локацій має знак стабільності, що показує можливість появи на ній порталу у інший світ, два знаки, що показують які пригоди можуть трапитися з гравцями на цьому місці. Деякі локації мають альтернативну пригоду, яку гравці можуть вибрати замість нормальної;
  - 1 таємну локацію Inner Sanctum, доступ до якої мають гравці у локації Silver Twilight Lodge за наявності Silver Twilight Lodge Membership;
  - Чорні, білі на чорно-білі стрілки між локаціями на вулицями, що показують куди будуть переміщатся чудовиська кожен хід гри;
  - Terror track, що показує, наскільки жителі міста налякані тим, що відбувається у місті;
  - Околиці міста, що служить як місце для чудовиськ, що не можуть перебувати у місті на даний момент через переповненість міста іншими чудовиськами;

- Небо, що служить як спеціальне місце для чудовиськ, які вміють літати і дозволяє переміщення у будь-яку локацію міста;
- Місце для гравців, втрачених у просторі на часі, щоб показати, що вони поза межами гри;
- Вісім інших світів, кожен з яких маркований двома або чотирма кольорами для позначення типів пригод у цих світах

- 16-и карт дослідників (одна з яких зображена на рис. 3.2), у ролі яких виступають гравці. Кожен/-а з дослідників має описаними:
  - локацію, на якій дослідник починає гру;
  - максимальну кількість розсудливості та витривалості.
  - три повзунка вмінь, які гравці можуть переміщувати на початку кожного ходу відповідно до своєї сфокусованості. При цьому кожен з повзунків зв’язує два вміння, одне з яких збільшується зліва-направо і один, що зменшується зліва-направо;
  - власне сфокусованість;
  - закріплені за дослідником предмети;
  - предмети, что дослідник отримує випадково на початку гри;
  - унікальне вміння.
- 28-ми типів чудовиськ (одне з яких зображено на рис. 3.3 і рис. 3.4), кожне з яких має:
  - тип переміщення, що описує переміщення чудовиська по місту і буває: нормальним (чорний колір рамки), стаціонарним (жовтий колір, не рухається взагалі), швидким (червоний колір, рухається двічі), літаючим (голубий колір, якщо чудовисько не може знайти ціль, то воно переміщується до Неба, що дає можливість йому атакувати будь-якого дослідника на вулицях міста) та унікальним (зелений колір);

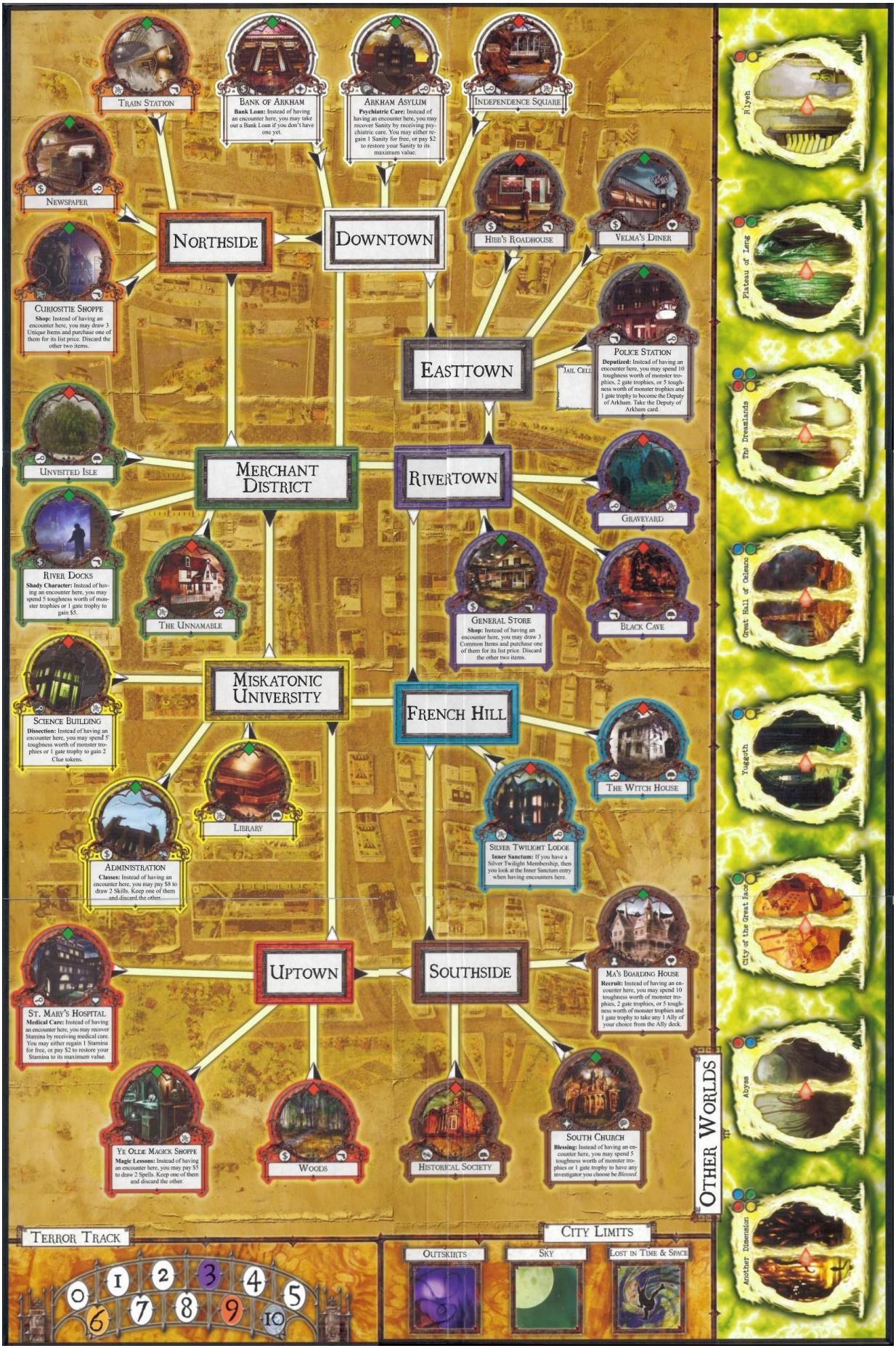

Рис 3.1 Дошка настільної гри

Рис. 3.2 Карта дослідника Amanda Sharpe

Рис. 3.3 Передня сторона карти чудовиська Byakhee

Рис. 3.4 Задня сторона карти чудовиська Byakhee

- витривалість, що описує кількість пошкоджень, які йому треба завдати, щоб вбити;
- horror rating/damage – фактично Horror check для дослідника;

- combat rating/damage – фактично check для дослідника;
- усвідомленість – функція, що описує наскільки просто прошмигнути повз цього чудовиська непомітно;
- вимір – знак, що впливає на те, куди рухається це чудовисько.

- Вісім карт Стародавніх (одне з яких зображене на рис. 2.5), що описують головних монстрів цієї гри, з полями:
  - combat rating – функція, що описує наскільки важко зробити Combat check проти цього Стародавнього;
  - захист – можливості стосовно самозахисту від атак дослідників;
  - поклонники – деякі чудовиська сильніші під керівництвом цього Стародавнього;
  - сила – потужне вміння цього Стародавнього, що на рівні з поклонниками ускладнює життя дослідникам;
  - атака – те, як дослідники будуть битися проти цього чудовиська, якщо воно все ж прокинеться;
  - doom track – візуалізація кількості знакі погибелі, що потрібні для пробуждення Стародавнього.
- 8 порталів у інші світи (один з яких зображений на рис. 2.6 та рис 2.7), включають у себе:
  - опис іншого світу, до якого веде портал;
  - функція, що описує складність закриття цього порталу;
  - символ виміру, до якого належить цей портал. Після закриття одного з порталів усі монстри на карті з цим самим символом виходять з гри.
- 88 карт (одна з яких зображена на рис. 2.8 та рис 2.9), які можуть тримати дослідники, розділені на:
  - 22 типи звичайних предметів (жовтого кольору);
  - 11 типів заклинань (синього кольору);
  - 26 типів унікальних предметів (червоного кольору);
  - 11 типів помічників (оранжевого кольору);

- 10 типів карт-вмінь (кольору вижовтівшої бумаги);
- 8 типів спеціальних карт (білого кольору).

Рис. 3.5 Карта Стародавнього Cthulhu

Рис. 3.6 Передня сторона порталу, що веде до іншого світу Another Dimension

Рис 3.7 Задня сторона будь-якого порталу

- ~7 пригод для кожної з локацій, об’єднані по три у карти пригод для кожного з районів міста;
- 48 карт-пригод одного з чотирьох кольорів (червоний, жовтий, синій, зелений) для інших світів, а також одна карта білого кольору, що інструктує перетасувати усю колоду цих карт.

Рис. 2.6 Передня сторона звичайної карти Ахе без текстової складової

Рис. 2.7 Задня сторона будь-якої звичаної карти

3.3 Типи ефектів

У даній грі усі ефекти можна розділити на два типи:

- Пасивні ефекти впливають на виконання деяких дій гравцями. Наприклад, карта Shotgun говорить, що, якщо вона використовується як зброя, то при підбрасуванні кісток усі шестірки вважаються двома успіхами;
- Активні ефекти, що або потребують дій з боку гравців для активації, або активуються автоматично під час певних подій. Наприклад, карти-

пригоди активуються під час відповідної фази і майже завжди є обов’язковими для виконання.

3.3.1 Активні ефекти, що описують дії дослідників

Ефект	Опис
...nothing happens.	Нічого не відбувається
...make a [skill] [modifier] [difficulty] check	Робить перевірку вказаного вміння, змінюючи кількість кісток відповідно до наданої функції модифікатора та складності перевірки
You or another investigator gains [token] ...	Дослідник обирає, кому він бажає передати відповідних знаків
... take the [card] ...	Дослідник бере карту з колоди, якщо подібна карта існує у відповідній колоді
... draw N [deck] ...	Дослідник бере N карт з зазначеної колоди
... roll a die and [action] ...	Дослідник підбрасує кістку і виконує дію відповідно до значення на кістці
... roll N dice. For every success rolled, [action1]. For every failure rolled, [action2].	Дослідник підбрасує N кісток. Перша дія виконується відповідно до кількості успішних кісток, друга – відповідно до кількості неуспішних
... immediately restore your Sanity to its maximum.	Дослідник відновлює свою розсудливість до максимуму
... immediately restore your Stamina to its maximum.	Дослідник відновлює свою витривалість до максимуму
... gain N [token] ...	Дослідник отримує N описаних знаків
... lose N [stamina] ...	Дослідник втрачає N описаних знаків
... spend N [stamina] ...	Те саме, що і у попередньому варіанті
... discard N cards ...	Дослідник обирає N карт зі своєї руки і переміщує їх до відповідних колод.

... gain N points, split between your [tokens] however you like	Дослідник розділяє N знаків між декількома групами знаків
... you may buy it for [modifier] than the list price	Дослідник може купити предмет за кількість грошей, що відповідає ціні цього предмету з приміненим модифікатором
Search the [deck] and purchase any one item of your choice at list price.	Дослідник може вибрати будь-яку карту з колоди і купити її за відповідну ціну.
Draw N [cardtype]. You may take M of them for free	Дослідник бере N карт і обирає M з них, повертаючи інші назад до відповідних колод
You may look at the top cards of the [decks]. You may purchase N at the list price.	Дослідник бере по одній карті з кожної колоди і може купити до N з обраних карт. Усі інші карти повертаются до відповідних колод
Search the [deck] and take any card you want	Дослідник бере будь-яку карту з колоди
Take the first [itemtype] from the [deck]	Дослідник бере першу карту-предмет з колоди.
... lose all of your money	Дослідник втрачає усі гроші
Donate either half your money (rounded up) ...	Дослідник втрачає половину грошей, округлену догори
Donate half your items (your choice, rounded up) ...	Дослідник переміщає половину своїх карт до відповідних колод, кількість округлена догори
Donate half your items (your choice, rounded down) ...	Дослідник переміщає половину своїх карт до відповідних колод, кількість округлена донизу
You must discard one of the following (your choice):	Дослідник втрачає вибрану кількість відповідних знаків/карт зі списку опцій

[type], ...	
... discard all of your [type]	Дослідник втрачає усі карти відповідного типу
... discard all of your hand.	Дослідник переміщує усі карти зі своєї руки і переміщує їх до відповідних колод
... remove N doom tokens	Дослідник вбирає N знаків погибелі з карти Стародавнього

3.3.2 Пасивні карткові ефекти (бонуси)

Бонус	Опис
+N [skill] check	Добавляє N нових кісток під час перевірки вказаного вміння
+N combat check (+M instead if your other hand is empty)	Добавляє N нових кісток під час Combat перевірки, або M, якщо зброєю зайнята тільки одна рука
+N combat check (+M if Opponent is [ability])	Добавляє N нових кісток під час Combat перевірки, або M, якщо опонент має відповідну здібність
+N Maximum Stamina	Дослідник має N більше витривалості
+N Maximum Sanity	Дослідник має N більше розсудливості
When using in combat, all rolled 6’s count as 2 successes	Якщо ця зброя використовується під час бою, усі кістки, результат яких шість, вважаються двума успіхами
You are Blessed	Під час перевірок, значення на кістках 4, 5 і 6 вважаються успішними
You are Cursed	Під час перевірок, тільки значення на кістках 6 вважається успішними
When you return to Arkham from an Other World, you can return to any location with an open gate, not just those leading	Під час повернення з іншого світу можна повернутися до будь-якого порталу

to the Other World you were in.	
If you are in Arkham, you may move to any street area or location in Arkham instead of your normal movement.	Під час Movement Phase можна переміститися до будь-якого другого іншого місця замість нормального переміщення
Whenever you have an encounter at the Silver Twilight Lodge, you have an encounter at the Inner Sanctum instead.	Доступ до локації Inner Sanctum
Your attacks are not affected by [ability].	Відповідна здібність будь-якого монстра не має впливу на дослідника, що має цю карту.
When you spend a Clue token to add to any [skill] check, add one extra bonus die.	Якщо дослідник витрачає знаки Доказу для того, щоб добавити кістку, добавляє ще одну кістку.

3.1.3 Активні карткові ефекти

Ефект	Опис
... immediately return to Arkham from an Other World	Дослідник, що знаходиться у Іншому Світі повертається у будь-яке місце у місті
... gain +N to all skill checks for the rest of this turn.	Дослідник отримує бонус у розмірі N кісток до будь-яких перевірок під час цього ходу
... make one Physical weapon a Magical weapon until the end of this combat	Змінює тип зброї з фізичної на магічну до кінця бою
... exhaust ...	Активний ефект карти не можливо використовувати до наступного ходу
... discard this ...	Карта переміщається до відповідної колоди
... return this card to the box ...	Карта покидає гру
... put one [token] on this card. If	Дослідник кладе відповідний знак на карту.

there are N [token] on this card, [action].	Якщо кількість знаків на карті N, відбувається дія.

Також деякі слова обмежують час використання карти:

Ефект	Під час	Опис
Upkeep:	Фази Upkeep	Якщо карта не стверждує зворотнього, вона **має** бути розіграна під час цієї фази
Movement:	Фази Movement	Карта може бути розіграна під час фази
[action] when [ally] joins you	Отримання карти	Карта виконує деяку дію, коли вона потрапляє до руки гравця (обмін картами між гравцями не входить в це правило)
[action] before making a [skill] check	Перед перевіркою	Карту можна розіграти перед виконанням перевірки.
[action] after failing a [skill] check	Після проваленої перевірки	Карту можна розіграти, якщо перевірка не набрала необхідної кількості успіхів
[action] at the end of each combat	Після бою	Карту можна розіграти після бою
[action] whenever you return to Arkham	Після повернення до міста	Карту можна розіграти по поверненню до міста
[action1] after drawing [drawable] to draw a new card in its place	Витягування типу карт	Під час витягування відповідного елементу з відповідної колоди, виконати потрібну дію для повторної спроби
When sealing a gate, ...	Закриття порталів	Карту можна розіграти під час закриття порталів

До відповідних слів можлива наявність деяких специфічних ефектів, що можуть відбуватися тільки у заданому контексті:

Ефект	Під час	Опис
... automatically succeed ...	Перед перевіркою	Автоматична успішна перевірка
... defeat all monsters in the current area	Перед перевіркою	Перемагає усіх монстрів на позиції дослідника
... to re-roll N di[c]e ...	Після проваленої перевірки	Знову підбрасує N неуспіших кісток
... to re-roll ...	Після проваленої перевірки	Знову підбрасує усі кістки
... to reduce monster's Combat damage to N Stamina ...	Після проваленої перевірки	Зменшує атаку монстра до N витривалості
... to avoid losing N Stamina/Sanity ...	Під час отримання урону	Виконує дію, зменшуючи урон нанесений ворогом на N

Також частина карт є:

Тип	Можливі дії	Опис
Заклинання	... cast this spell to [action] ...	Дослідник проходить Spell check. Якщо перевірка успішна, виконується відповідна дія
Зброя	... lower a monster's toughness by N (to a minimum of M)	Зменшення витривалості монстра на N (до мінімума у M)
	... ignore one of its special abilities other than [ability]	Дослідник обирає, яке з вмінь монстра не працює (окрім описаного вміння)

	... to pass one Combat Check.	Дослідник проходить цю перевірку

3.1.4 Ефекти карт-пригод у місті

Ефект	Опис
A gate appears!	На позиції дослідника відкривається портал і дослідник поглинається ним
A monster appears!	На позиції дослідника з’являється монстр, з яким дослідник має провести бій
You doze off and enter the [otherworld]. Have an encounter there, then immediately return here	Дослідник переміщується до іншого світу, бере карту-пригоду там, виконує дії на ній та повертається назад до міста
... lose your next turn	Дослідник втрачає можливість робити будь-що на цілий хід
... you are arrested	Дослідник переміщується до Police Station Jail
... you are Lost in time and space	Дослідник втрачений у часі та просторі
Move to the street	Дослідник переміщується до вулиць відповідного району міста
Stay here next turn	Дослідник втрачає можливість переміщення на один хід
Turn the top card of one location deck of your choice face up. The next investigator to have an encounter at that location draws that encounter card.	Дослідник перегортає верхню карту колоди пригод однієї з локацій.
... claim any one monster	Дослідник забирає будь-якого монстра зі стола, як

on the board as a trophy	трофей
Draw a monster from the cup and take it as a monster trophy	Дослідник бере наступного монстра та забирає його, як трофей
Roll a die for each open gate one at a time. On a success the gate is closed	Дослідник підбрасує кількість кісток рівну до кількості відкритих порталів на столі. Кожен успіх закриває відповідний портал.
... draw again for a different encounter.	Дослідник бере нову карту-пригоду
You may choose another investigator and bring it into play as a new character	Гравець може змінити дослідника, втрачаючи усі
... close one gate of your choice	Дослідник закриває один з порталів на вибір
Move to [location]...	Дослідник переміщується до вказаної локації
Move to any location or street area in Arkham.	Дослідник переміщується до обраної локації
You may sell any of [cardtype] for twice its listed price.	Дослідник може продати будь-яку кількість карт описаного типу відповідно до ціни кожної з них.
If you are not reduced to 0 Sanity, [action] ...	Якщо дослідник ще має деяку кількість розсудливості, то виконується дія

3.1.5 Ефекти карт-пригод у Інших Світах

Roll 1 die for each point of [token] you have. Lose 1 [token] for each die that is not a success, [action] for each die that did.	Дослідник підбрасує кількість кісток, рівну кількості його певних знаків, втрачаючи відповідні знаки при невдачі та виконуючі дію на кожну вдалу кістку

If you are in the first area of [otherworld], move to the second area. If you are in the second area, return to Arkham.	Дослідник переміщується вперед в межах іншого світу. Якщо дослідник був у другій частині іншого світу, він/вона повертається назад до міста.
... return to Arkham	Дослідник повертається до міста
... you are Lost in time and space	Дослідник втрачений у часі та просторі
Stay here next turn	Дослідник втрачає можливість переміщення на один хід
Draw a monster from the cup and take it as a monster trophy	Дослідник бере наступного монстра та забирає його, як трофей
... you are devoured.	Дослідник зжертий і гравець має вибрати нового дослідника
A monster appears!	На позиції дослідника з’являється монстр, з яким дослідник має провести бій
... close one gate of your choice	Дослідник закриває один з порталів на вибір

3.1.6 Активні ефекти карт-міфів

Every player [action] ...	Кожен гравець виконує дію
First player [action] ...	Перший гравець виконує дію
All monsters in the [location] are returned to the cup.	Усі монстри у описаних позиціях на карті повертаються до колоди.
... releasing N monsters into [location]	У відповідній локації з’являється N нових монстрів
Any [monster] previously claimed as monster	Дослідники втрачають усі трофеї відповідного монстра і кожен з цих монстрів з’являється у

trophies by players return to life and are placed in the [location].	відповідній локації
All investigators in jail are released	Усі дослідники, що зараз знаходяться у Police Station Jail, вільні
All investigators currently lost in time and space immediately return to Arkham	Усі дослідники, втрачені у часі та просторі, повертаються до будь-якої позиції у місті, на їх вибір.

3.1.7 Пасивні ефекти карт-міфів

Gates cannot be sealed, although they can still be closed.	Дослідники не можуть повноцінно запечатувати ворота, але все ще можуть закривати їх
It costs N fewer Clue tokens to seal gates.	Запечатування коштує на N знаки доказів менше
... all Spells have a Sanity cost of 0.	Заклинання не потребують витрат розсудливості для використання.
... no spells may be cast	Заклинання не можна використовувати
Investigators cannot gain [Sanity/Stamina], except by [action]	Дослідники не можуть отримувати розсудливість або витривалість, окрім описаних способів
[monster] have their toughness increased by N	Витривалість описаних монстрів збільшена на N
The difficulty to seal or close gates to [otherworld] is increased by N	Запечатування або закриття воріт до описаного Іншого Світу має підвищену складність на N
All [ability] monsters have their toughness	Витривалість монстрів з описаним вмінням збільшена на N

increased by N	
Investigators who end their movement in [location] ...	Дослідники, що завершили фазу Movement у описанії локації виконують дію.
[monster] do not move	Описані монстри не можуть рухатися
Fast Monsters move like normal monsters.	Швидкі монстри рухаються, як нормальні
If a [monster] enters play, return it to the cup and draw a different monster.	Описані монстри не можуть з’являтися на столі і відповідно замінюються іншим монстром при спробі це зробити
... the terror level cannot increase	Рівень страху у місті не може збільшуватися
Roll a die at the end of every [phase] while this card is in play. On M, [action]	В кінці описаної фази підкидається кістка і якщо її значення дорівнює M, виконується дія

Висновки до розділу 3

У цьому розділі мною були згруповані усі важливі елементи гри та усі ключові слова ефектів, що використовуються картами, картами-пригодами та картами-мифами. Таке розділення зробить можливе створення потужних типів, об’єднаних у ациклічні направлені графи, що в свою чергу дозволить ефективний опис елементів гри у рамках обраної мови програмування.

Слід зазначити, що незважаючи на повних опис усіх типів ефектів, у кінцевому коді усе одно будуть наявні розбіжності, адже слова мов не співвідносяться до імплементацій одне до одного, відповідно прийдется створити дещо більше підтипів, що будуть синонимічні, для підвищення читаємості коду програми.

Розділ 4.
Розробка гри

4.1 Опис типів

Оскільки у даній грі ми зберігаємо лише базові дані і історію всього світу, опис типів є достатньо тривіальним, адже нам треба описати тільки стартові стани кожного з типів.

4.1.1 Звичайні карти

Кожна карта має ім’я та ідентифікатов, оскільки кожен гравець може мати декілька карт і відповідно для швидкого порівняння двух карт нам потрібні лише ці два поля. Поле bonus описує список пасивних елементів карти, поле effect – активних. Поле kind каскадно описує тип карти і усі приналежні підтипи (наприклад, чи є ця карта предметом і якщо так, чи є вона зброєю).

Опис таких карт спеціальним типом дозволить вважати картами, що тримає дослідник тільки карти, які він може тримати, що, безумовно, є плюсом.

```
data Card = Card
            { _name :: Text
            , _id :: Int
            , _bonus :: [Bonus]
            , _effect :: [Effect]
            , _kind :: Kind
            }
```

4.1.2 Локації

Оскільки локації бувають трьох типів (вулиці, таємні локації і саме локації), то і зберігати їх потрібно у відповідних конструкторах типів. У той же час:

- для вулиць нам потрібно знати лише район (кожен район має тільки одну вулицю), у котрому вони знаходяться і вулиці, до яких можна перейти з цієї вулиці;
- єдина таємна локація фактично знаходиться на місці іншої локації, що означає, що її не треба описувати;
- локації мають унікальні назви, що і буде служити способом їх порівняння, але навідміну від вулиць також можуть бути нестабільними, закриватися при певному рівні страху у місті та мати спеціальну пригоду.

```
data Location = Street
                  { _neighborhood :: Neighborhood
                  , _near :: [Neighborhood]
                  }
              | Location
                  { _name :: Text
                  , _neighborhood :: Neighborhood
                  , _stability :: Stability
                  , _terror :: Maybe Int
                  , _special :: Maybe Encounter
                  }
```

4.1.3 Дослідники

Кожен дослідник має ім’я і дослідники унікальні, що означає, що їх можна порівнювати лише за ім’ям. Кожен з них також має своє місцезнаходження на початок гри, початкові стани витривалості, розсудливості, фокусу, грошей, знаків доказу, унікальне вміння та деякі предмети, що надаються їм перед їх першим ходом.

Варто відзначити наявність типу `Skills`, що описує базові значення усіх повзунків вмінь, адже всі дослідники починають гру з крайньої лівої позиції.

```
data Investigator = Investigator
                    { _name :: Text
                    , _home :: Location
                    , _sanity :: Int
                    , _stamina :: Int
                    , _focus :: Int
                    , _skills :: Skills
                    , _money :: Int
                    , _clues :: Int
                    , _possessions :: [Possession]
                    , _ability :: Ability
                    }
```

4.1.4 Монстри

Монстри поділяються на два типи: звичайні та маски. Звичайні монстри потребують також ідентифікатор, у той час як маски унікальні. Усі монстри можуть мати деякий Horror та Combat rating, відносно яких дослідники роблять перевірки під час зустрічі з монстром, також

витривалість, тип переміщення, вимір, до якого цей монстр належить та одне або декілька вмінь.

```
data Monster = Monster
                { _name :: Text
                , _id :: Int
                , _horror :: Maybe Horror
                , _toughness :: Int
                , _combat :: Maybe Combat
                , _awareness :: Modifier
                , _movement :: Movement
                , _dimension :: Dimension
                , _ability :: [MonAbility]
                }
             | Mask
                { _name :: Text
                , _horror :: Maybe Horror
                , _toughness :: Int
                , _combat :: Maybe Combat
                , _awareness :: Modifier
                , _movement :: Movement
                , _dimension :: Dimension
                , _ability :: [MonAbility]
                }
```

4.1.5 Карти-міфи

Усі картки-міфи поділяються на три підтипи: заголовки газет, стан навколишнього середовища та чутки. Фактично вони усі мають одну й ту ж

саму структуру, єдиною різницею є те, що заголовки газет майже не мають постійних ефектів, стани навколишнього середовища знаходяться у грі до наступного стану, а чутки знаходяться у грі до вирішення їх ефектів гравцями або самою грою.

Через наявність виключень зі сторони як заколовків газет, так і станів навколишнього середовища, усі карти міфів мають і миттєві, і постійні ефекти, що не дає можливості створити абсолютно точну систему типів саме для цієї ситуації.

Кожна карта має унікальне ім’я, що і відрізняє її.

```
data Mythos = Headline
                { _name :: Text
                , _gate :: Location
                , _clue :: Location
                , _white :: [Dimension]
                , _black :: [Dimension]
                , _constant :: [Constant]
                , _instant :: Instant
                }
            | Environment
                { _name :: Text
                , _gate :: Location
                , _clue :: Location
                , _white :: [Dimension]
                , _black :: [Dimension]
                , _constant :: [Constant]
                , _instant :: Instant
                }
```

```
        | Rumor
            { _name :: Text
            , _gate :: Location
            , _white :: [Dimension]
            , _black :: [Dimension]
            , _state :: Maybe RumorState
            , _constant :: [Constant]
            , _instant :: Instant
            , _pass :: Condition
            , _fail :: Condition
            }
```

4.1.6 Стародавнє

Кожне Стародавнє має унікальне ім’я, що і буде служити для їх швидкого порівняння, модифікатор, пасивне вміння, що зазвичай збільшує силу монстрів у місті, можливо має ефект на початку гри, або під час пробудження та також має спеціальну атаку, яку воно використовує під час фази атаки Стародавнього під час фінального бою.

```
data Ancient = Ancient
                { _name :: Text
                , _combatRating :: Modifier
                , _passive :: [Passive]
                , _starting :: Once
                , _wakeup :: Once
                , _attack :: Active
                }
```

4.1.7 Інші світи

Кожен з Інших Світів має своє унікальне ім’я та деякий набір кольорів карт, що мають пригоди для цього іншого світу.

```
data OtherWorld = OtherWorld
                  { _name :: Text
                  , _colors :: [Color]
                  }
```

4.1.8 Портали

Кожен портал має пов’язаний з ним вимір, Інший Світ, та модифікатор, що описує, наскільки важно його закрити. У порталів немає імен, оскільки комбінація цих трьох елементів фактично є унікальним для кожного з порталів.

```
data Gate = Gate
            { _dimension :: Dimension
            , _otherWorld :: OtherWorld
            , _modifier :: Modifier
            }
```

4.1.9 Місто Arkham

Відповідно початковий стан міста має у собі деякий набір імплементацій усіх вищеописаних типів, а також колод пригод для обох світів. Варто зауважити, що відповідно до використовуваного типу програмування цей тип має мати абсолютно усі карти, що мали би бути на столі на початку гри, що означає, що, наприклад, п’ять монстрів-масов

Стародавнього Nyarlathotep в усіх іграх, де Стародавнім не є Nyarlathotep не будуть частиною гри, але все одно будуть частиною стану.

```
data Arkham = Arkham
                { _investigators :: [Investigator]
                , _locations :: [Location]
                , _encounters :: [Encounters]
                , _otherWorlds :: [OtherWorld]
                , _otherWorldEncounters
                    :: [OWEncounters]
                , _gates :: [Gate]
                , _cards :: [Card]
                , _monsters :: [Monster]
                , _ancient :: Ancient
                , _mythos :: [Mythos]
                }
```

4.2 Реалізація ефектів

Технічно існує декілька способів реалізації ефектів, кожен з яких має свої переваги та недоліки. Відповідно, можливо:

1. Описати усі активні ефекти одним типом та усі пасивні ефекти одним типом. У такому варіанті потрібно використовувати фантомні типи для забезпечення безпеки на рівні типів, адже різні групи елементів гри не можуть виконувати абсолютно усі ефекти.
2. Описати кожен ефект окремим деревом для кожного елемента, що їх має. На практиці означає прирівнювання функцій примінення ефектів

одного елемента до функцій іншого, що може у деякій степені створити хаос при невірному розташуванні модулів.

На практиці немає абсолютно вірного способу імплементацій дерева типів, адже чисте використання першого методу створює конфуз на рівні опису типів, а другого – на рівні імплементації.

Я обрав другий метод для усього проекту, з використанням першого для карт, адже у карт є ділення на фази з ключовими словами, що не можуть використовуватися у невірному контексті. Відповідно:

```
data Scope = Other
           | Ancient
           | Weapon
           | BeforeCheck
           | OnFailedCheck
           | WhenLosing


type family Needs (a :: Scope) (s :: Scope) where
  Needs a a = '() ~ '()
  Needs a s = TypeError
                ( Text "Required scope "
                    :<>: ShowType a
                    :<>: Text " does not match given "
                    :<>: ShowType s )
```

Сімейство типів `Needs` описує типи, що відповідають заданим, відповідно ефект знищення усіх монстрів на позиції дослідника, що сам по собі може виконуватися тільки перед перевіркою має вигляд

```
DefeatArea :: Needs BeforeCheck s => Eff s,
```

де `Eff` – тип, контруктором якого є `DefeatArea`

4.3 Подальша імплементація

Для опису взаємодій нам знадобляться:

- Клас низькорівневих взаємодій, що зробить можливим різні імплементаційні описи базових функцій, що відрізняються для клієнта та сервера даної гри;
- Монади Reader та State, як частина бібліотеки mtl [7] для зручного опису функцій з унеможливленням зміні станів, на яких базується поточний стан;
- Декілька вхідних та вихідних потоків для забезпечення комунікації між сервером та клієнтом, а також між даною частиною програми та графічним інтерфейсом користувача, адже графічний інтерфейс розглядає світ не як набір станів, а як набір модифікацій даних, що вочевидь не підходить під описану систему. Відповідно для створення графічних інтерфейсів підходить концепція реактивного функціонального програмування [8], зокрема реалізована у мові Haskell такими бібліотеками, як reflex [9] та reactive-banana [10].

Висновки до розділу 4

У даному розділі мною були імплементовані головні типи гри Arkham Horror, описаний обраний мною шлях імплементації ефектів з підвищеною безпекою на рівні типів та надані ідеї подальшої розробки цієї роботи.

Дана імплементація не є фактично повною, оскільки розмір навіть базової гри кремезний: більше 500 елементів, що унеможливлює повноцінну реалізацію цього проекту за час та простір, відведені на це дипломом. Але, незважаючи на це, видно, що змінена структура програми дає чіткі плюси під час реалізації, адже групування ефектів та чітка поліморфна система типів мови Haskell істотно спрощує написання програми та її читання.

Висновки

У даній дипломній роботі мною були розглянути існуючі програмно-реалізовані настільні ігри, яких виявилося дуже мало через нерозповсюдженість подібних реалізацій у індустрії на даний час та встановлено, що на даний момент їх реалізація є не зовсім оптимальною.

У спробах реалізувати настільну гру 2005го року Arkham Horror, мною були проаналізовані сучасний об’єкто-орієнтований підхід та майже не використовуваний чисто функціональний підхід, і останній був обраний мною як ідейна платформа для реалізації цієї роботи.

Усі типи та ефекти гри Arkham Horror були мною проаналізовані, спрощені та згруповані для лаконічності репрезентації у вибраною мною мові Haskell, що також дозволила мені добавити деяку безпеку на рівні типів. У той же час очевидно, що мова Haskell не дає достатньої кількості засобів для програмуванні типів на даний момент, що унеможливлює частину підходів.

Список використаних джерел

1. Arkham Horror Wiki [Електронний ресурс]. http://www.arkhamhorrorwiki.com/Main_Page (дата звернення 27.05.2019)
2. Fantasy Flight Games [Електронний ресурс]. https://www.fantasyflightgames.com/en/index/ (дата звернення 27.05.2019)
3. Hearthstone [Електронний ресурс]. Режим доступу: https://playhearthstone.com/en-us/ (дата звернення 27.05.2019)
4. Arkham Horror Video Game | Wrangling the Ancient Ones with code [Електронний ресурс]. Режим доступу: https://arkhamhorrorvideogame.wordpress.com/ (дата звернення 27.05.2019)
5. Download Microsoft XNA Framework Redistributable 4.0 from Official Microsoft Download Center [Електронний ресурс]. https://www.microsoft.com/en-us/download/details.aspx?id=20914 (дата звернення 27.05.2019)
6. Haskell Language [Електронний ресурс]. https://www.haskell.org/ (дата звернення 27.05.2019)
7. mtl: Monad classes, using functional dependencies [Електронний ресурс]. http://hackage.haskell.org/package/mtl (дата звернення 27.05.2019)
8. Functional Reactive Programming [Електронний ресурс]. https://en.wikipedia.org/wiki/Functional_reactive_programming (дата звернення 27.05.2019)
9. Reflex FRP [Електронний ресурс]. https://reflex-frp.org/ (дата звернення 27.05.2019)
10. reactive-banana: Library for functional reactive programming (FRP). [Електронний ресурс]. http://hackage.haskell.org/package/reactive-banana (дата звернення 27.05.2019)

Додатки